
# The Relational Data Model

Instructor: David Martin

*Reference:*
*A First Course in Database Systems, 3rd edition, Chapter 2.1, 2.2*

# Outline

- Quick review
  - Database
  - Database management system (DBMS)
- Relational data model
  - Relations & their parts  (informally)
    - Basic terminology
  - Relations & their parts (more formally)
  - Key Constraints (Keys)
- Some relevant bits of discrete mathematics

# What is a Database?

- A *database* is an organized, persistent collection of data.
  - Stored digitally.
  - Managed by a Database Management System (DBMS).

# What is a Database Management System?

A *database management system (DBMS)* is a software system that assists in creating, maintaining, and using (typically large) datasets effectively & efficiently.

Key functionalities:

- Creation
  - Specification of *schemas* (descriptions of logical structure of data)
- Querying & modification of data
  - Asking questions to retrieve data
  - Adding, deleting & changing data

  Using a high-level language
- Storage Management
  - Long-term, scalable use of computer memory & storage resources
- Durability
- Access control & integrity

# More Key Characteristics of a DBMS

- Data Model
  - Provides an abstraction of the underlying data
  - Provides organizing principles for DBMS capabilities
  - Provides theoretical basis
- High-level language for managing data
  - For defining, updating, retrieving and processing data
- Transaction Processing
  - Concurrent access and updates, crash recovery
- Performance
  - Response time/latency
  - Throughput
  - Scalability

# Lecture Schedule

## Initial Draft, subject to change

| | Lectures | Dates | Chapters |
|---|---|---|---|
| **History and Introduction** | 1 | 6/27 | 1 |
| **The Relational Data Model** | 1 | 6/29 | 2.1,2.2 |
| **SQL: DDL, DML; Defining relations and constraints, writing queries;<br>Modifications, transactions;<br>Views, indexes** | 4 | 7/6-7/18 | 2.3, 2.5*; 6.1-6.7; 8.1-8.4<br>(except 8.2.3 and 8.4.3) |
| **Relational Algebra** | 3 | 7/20-7/27 | 2.4, 2.5 |
| **Wrap up above topics; Midterm review (time permitting)** | 1 | 8/1 | - |
| **Midterm** | 1 | 8/3 | - |
| **Constraints and Triggers** | 1 | 8/8 | 7.1-7.3, 7.5 |
| **Database Application Development** | 1 | 8/10 | 9.1,9.2,9.6 |
| **Schema Refinement and Normal Forms** | 2 | 8/15-8/17 | 3.1-3.5<br>(except 3.2.5, 3.4.2, 3.4.3, 3.5.3) |
| **Semistructured Data Model: XML, JSON** | 2 | 8/22-8/24 | 11.1-11.3, 12.1-12.2 |
| **NOSQL; Knowledge Bases** | 1 | 8/29 | |
| **Final Exam** | 1 | 8/31 | |

* Textbook section 2.5: Focus on the concepts first time through; we will cover the notation in the Relational Algebra material


# The Relational Data Model

Instructor: David Martin

*Reference:*
*A First Course in Database Systems, 3rd edition, Chapter 2.1, 2.2*

# What is a Data Model?

- A *data model* is a conceptual formalism that provides:
  1. A notation for describing the *structure* of data
  2. A set of *operations* for querying & manipulating data
     - E.g., query, insert, update, delete
  3. A way of specifying *constraints* on the data

- We have mentioned (or will mention):
  - Network data model
  - Hierarchical data model
  - Relational data model
  - Semistructured data model
    - XML, JSON

# What is a relational data model?

- The relational data model (Edgar F. Codd, 1970)
  - Data is described and represented using tables
    - Which are based on the mathematical concept of a *relation*

- What is a "relation"?
  - A structure (table) with rows and columns (in database parlance)
  - A subset of a Cartesian product of sets (math definition; we'll return to this shortly)

# Relations

- A relation is a table, with rows (*tuples, records*) and columns

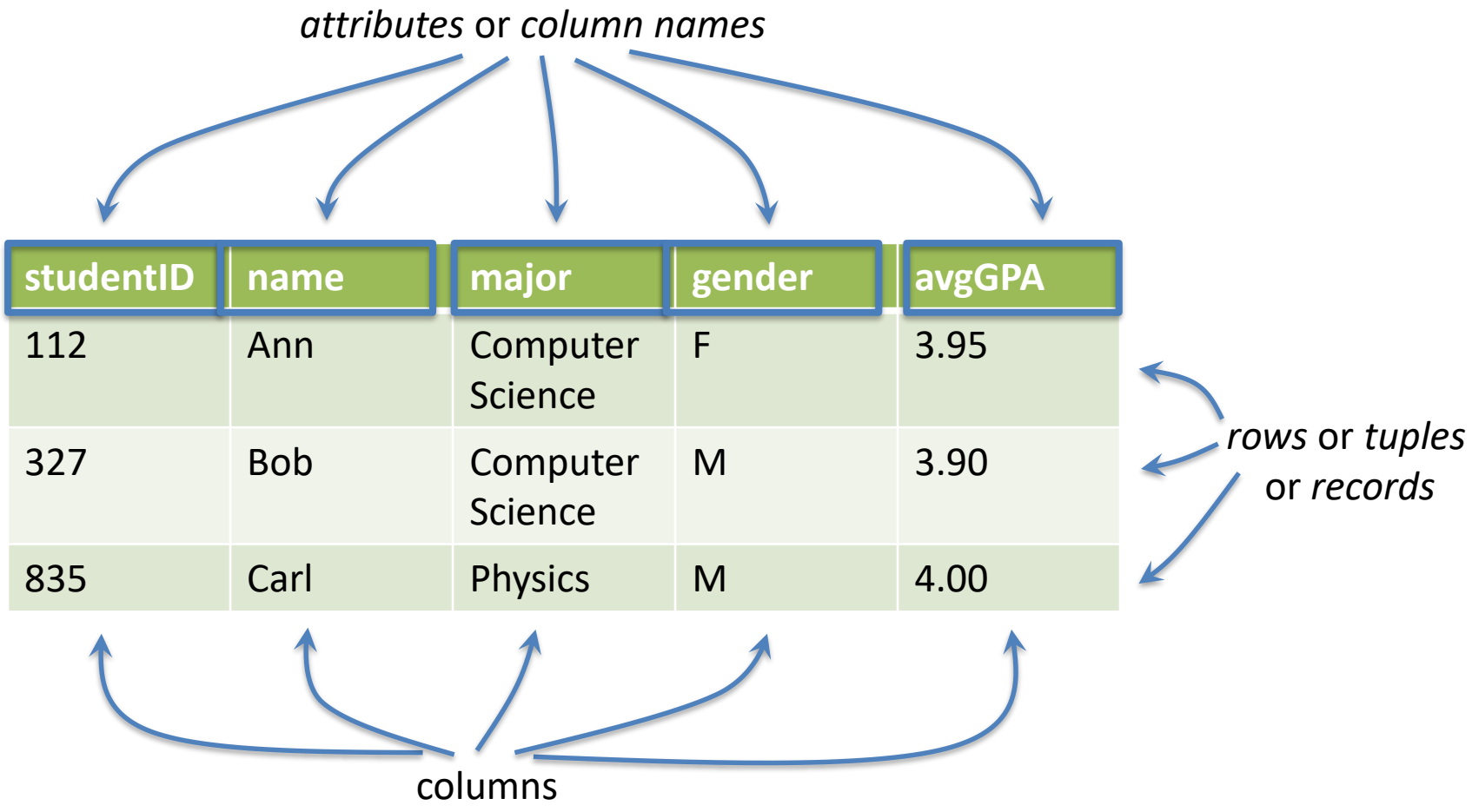
Student

| studentID | name | major | gender | avgGPA |
|---|---|---|---|---|
| 112 | Ann | Computer Science | F | 3.95 |
| 327 | Bob | Computer Science | M | 3.90 |
| 835 | Carl | Physics | M | 4.00 |

Course

| courseID | description | department |
|---|---|---|
| CMPS101 | Algo. | CS |
| BINF223 | Intro. to bio. | Biology |

Also: Teach, Enroll, Professor, …




We also talk about:
the *fields (*or *components)* of a tuple

3 rows, 5 columns. The relation has *arity* 5.

# Example of Relations

- Wall Street Stocks:
  - http://www1.nyse.com/about/listed/lc_ny_name_A.html?ListedComp=All
- Presidential info:
  - http://politicsandprosperity.com/facts-about-presidents/
- NFL Stadium info:
  - http://en.wikipedia.org/wiki/List_of_current_National_Football_League_stadiums
- National Basketball Association Standings
  - http://www.nba.com/standings/team_record_comparison/conferenceNew_Std_Div.html

# Attributes

- An *attribute* is the name of a column in a relation

Student

| studentID | name | major | gender | avgGPA |
|---|---|---|---|---|
| 112 | Ann | Computer Science | F | 3.95 |
| 327 | Bob | Computer Science | M | 3.90 |
| 835 | Carl | Physics | M | 4.00 |

Course

| courseID | description | department |
|---|---|---|
| CMPS101 | Algo. | CS |
| BINF223 | Intro. to bio. | Biology |

Also: Teach, Enroll, Professor, …

# Relation Schema

- A *relation schema* R is a set $\{A_1, \ldots, A_k\}$ of attributes, often written as $R(A_1, \ldots, A_k)$, where $A_i$ is the name of the *i*th column of the relation.
  - The *datatype/domain* of each attribute is an elementary type, such as integer, string or an enumerated type
    - not an object, array, list, or other compound structure
  - E.g., Student(studentID:int, name:str, major:str, gender:gender, avgGPA:double)
  - … or simply, Student(studentID, name, major, gender, avgGPA)

    when types are implicit

# Relation Schema

- A *relation schema* describes the logical structure of a relation
- Student(studentID:int, name:str, major:str, gender:gender, avgGPA:double)

| Student | studentID: int | name: str | major: str | gender: gender | avgGPA: double |
|---|---|---|---|---|---|
| | 112 | Ann | Computer Science | F | 3.95 |
| | 327 | Bob | Computer Science | M | 3.90 |
| | 835 | Carl | Physics | M | 4.00 |

# Domains

- The *domain* of an attribute is its datatype

Student

| studentID: | name: | major: | gender: | avgGPA: |
|---|---|---|---|---|
| int | str | str | gender | double |
| 112 | Ann | Computer Science | F | 3.95 |
| 327 | Bob | Computer Science | M | 3.90 |
| 835 | Carl | Physics | M | 4.00 |

# Tuples

- A *Tuple* in a relational DB is a Row
- Each tuple has a *value* (or NULL) for each *attribute*

Student

| studentID | name | major | gender | avgGPA |
|---|---|---|---|---|
| 112 | Ann | Computer Science | F | 3.95 |
| 327 | Bob | Computer Science | M | 3.90 |
| 835 | Carl | Physics | M | 4.00 |

Course

| courseID | description | department |
|---|---|---|
| CMPS101 | Algo. | CS |
| BINF223 | Intro. to bio. | Biology |

Also: Teach, Enroll, Professor, …

# Relation Schema and Instance of a Relation Schema

- An *instance of a relation schema* is a relation that conforms to the relation schema.
  - E.g., for this relation schema:
    Student(studentID:int, name:str, major:str, gender:gender, avgGPA:double)
    - Every tuple in the relation must be a 5-tuple.
    - The *i*th component of each tuple in the relation must have the corresponding type (correct domain).

this is an instance:

{ (112, “Ann”, “CS”, “F”,3.95),
(327, “Bob”, “CS”, “M”, 3.90),
(835, “Carl”, “Physics”, “M”, 4.00) }

✓

{ (“112”, “Ann”, “CS”, “F”,3.95),
(“327”, “Bob”, “CS”, “M”, 3.90),
(“835”, “Carl”, “Physics”, “M”, 4.00) }

✗

# A Relational Database Schema

- A *relational database schema* or, simply, a *database schema* is a set of relation schemas with disjoint relation names.

- A university database schema:
  - Student(studentID, name, major, gender, avgGPA)
  - Course(courseID, description, department)
  - Teach(profID, courseID, quarter, year)
  - Enroll(studentID, courseID, grade)
  - Professor(profID, name, department, level)

# Instance of a Database Schema

- An *instance of a database schema* $\{R_1, \ldots R_k\}$ (or a *database instance* in short) is a set $\{r_1, \ldots, r_k\}$ of relations such that $r_i$ is an instance of $R_i$, for $1 \leq i \leq k$.

Student

| studentID | name | major | gender | avgGPA |
|---|---|---|---|---|
| 112 | Ann | Computer Science | F | 3.95 |
| 327 | Bob | Computer Science | M | 3.90 |
| 835 | Carl | Physics | M | 4.00 |

Course

| courseID | description | department |
|---|---|---|
| CMPS101 | Algo. | CS |
| BINF223 | Intro. to bio. | Biology |

Also: Teach, Enroll, Professor, …

# First Normal Form (1NF)

- Domains in relational database were atomic. That atomicity means that relational databases satisfy what Ted Codd called "First Normal Form" 1NF.
  - E.g., Major has to be a single value, not a list of values
  - E.g., Address has to be a single value, not a structure
- Relational databases are "structured" and are in 1NF, which makes many things simple
  - Semi-structured and unstructured data (e.g., with XML, JSON or Protocol Buffers) doesn't satisfy 1NF.
  - Note: some relational database systems now also can include semi-structured data, not just structured data.

# What's The Idea Behind Keys?

Movies

| Title | Year | Length | Genre | studioName | producerC# |
|---|---|---|---|---|---|
| Pretty Woman | 1990 | 119 | Romantic | Disney | 999 |
| Monster's Inc. | 1990 | 121 | Animation | Dreamworks | 223 |
| Jurassic Park | 1998 | 145 | NULL | Disney | 675 |
| King Kong | 1933 | 100 | Adventure | RKO | 886 |
| King Kong | 1976 | 134 | Adventure | Paramount | 114 |
| King Kong | 2005 | 187 | Adventure | Universal | 338 |

# What's The Idea Behind Keys?

Movies

| Title | Year | Length | Genre | studioName | producerC# |
|---|---|---|---|---|---|
| Pretty Woman | 1990 | 119 | Romantic | Disney | 999 |
| Monster's Inc. | 1990 | 121 | Animation | Dreamworks | 223 |
| Jurassic Park | 1998 | 145 | NULL | Disney | 675 |
| King Kong | 1933 | 100 | Adventure | RKO | 886 |
| King Kong | 1976 | 134 | Adventure | Paramount | 114 |
| King Kong | 2005 | 187 | Adventure | Universal | 338 |

For what *set of attributes* are we guaranteed that every row will have a different combination of values on those attributes?

# Keys

Several different ways to say the same thing -
A key is a set of attributes for which:

- Every row will have a different combination of values on those attributes
- Every two distinct rows must differ in their values on those attributes
- For any given selection of values for those attributes, there can be at most one row having those values
- If two tuples agree on their values for those attributes, then they must be the same tuple

Keys are also *minimal*: see next slide.

# Keys (a little more formally)

- A *key constraint* (or a *key* in short) of a relation schema R is a subset K of attributes of R such that
  1. For every instance r of R,
     - No two tuples of r can have the same values in all the attributes of K
  2. *Minimal*: no proper subset of K has the above property.

# Primary Keys, SuperKeys

- The *primary key* is a key selected (by the DBMS administrator) for the DBMS to rely on (for indexing, query planning, optimizations, etc.)
  - There can only be one primary key for a table
  - However, there can be other keys, called *unique keys*, which we'll get to before long

- A *superkey* is a set of attributes of R that includes a key of R.
  - That is, for any superkey, some key is a subset of the attributes.
  - All keys are superkeys but some superkeys are not keys.

# Primary Keys, SuperKeys: Examples

- Student(studentID, name, address, major, gender, avgGPA).
  - {studentID} is a key. It is also a superkey.
  - {studentID, name} is a superkey but not a key.
  - {studentID, name, major, gender, avgGPA} is a superkey but not a key.

- Also:
  - {studentID} would be used as the primary key.
  - {name, address} is another key (and a superkey).
    - Note: This is a questionable toy example
  - {name, address, avgGPA} is a superkey

# Database Schema for Our Movies DB

- What attribute(s) would make a good key for each of these relations?

  Movies(title, year, length, genre, studioName, producerC#)

  StarsIn(movieTitle, movieYear, starName)

  MovieStar(name, address, gender, birthdate)

  MovieExec(name, address, cert#, netWorth)

  Studio(name, address, presC#)

# Database Schema for Our Movies DB

- Here are some reasonable answers; however -
  - Not necessarily the only reasonable answers
  - The keys given for Movies, StarsIn, and MovieStar are probably not good enough for a comprehensive, international database

Movies(<u>title</u>, <u>year</u>, length, genre, studioName, producerC#)

StarsIn(<u>movieTitle</u>, <u>movieYear</u>, <u>starName</u>)

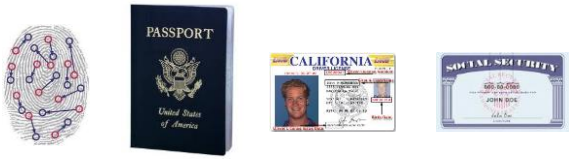
MovieStar(<u>name</u>, address, gender, birthdate)

MovieExec(name, address, <u>cert#</u>, netWorth)

Studio(<u>name</u>, address, presC#)

# Examples

- Student(studentID, name, major, gender, avgGPA).
  - {studentID} is a key. It is also a superkey.
  - {studentID, name} is a superkey but not a key.
  - {studentID, name, major, gender, avgGPA} is a superkey but not a key.




- There can be multiple keys in general.
  - One key is chosen and defined as the *primary key*
- Student(studentID, name, address, dob, major, gender, avgGPA)
  - {studentID}, {name, address} are keys and also superkeys.
    - {studentID} would normally be used as the primary key.
    - What about {name, dob}
    - What about {name, dob, avgGPA}
  - Can you think of a realistic multi-attribute key for a different table?

# Types of Relations in an RDB

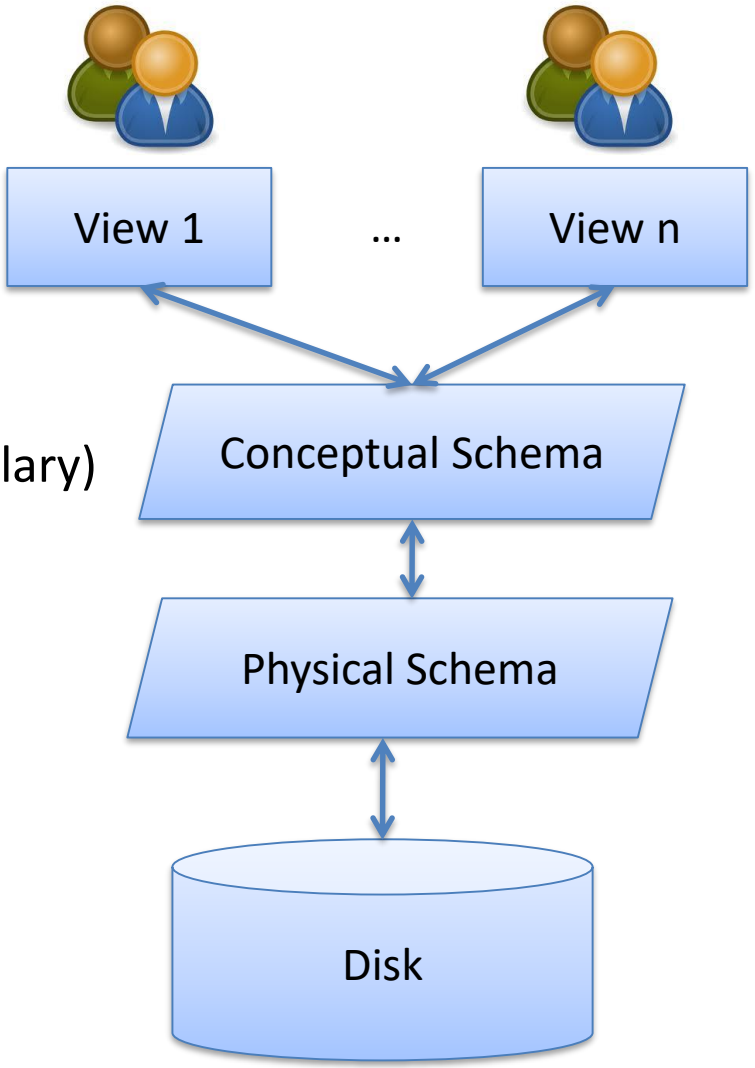
- Three types of relations
  1. Base relations (stored relations, tables)
     - These are tables that contain tuples and can be modified or queried.
  2. Views (derived relations, virtual relations)
     - Views are relations that are defined in terms of other relations but they are not stored. They are constructed only when needed.
  3. Temporary tables
     - These are tables (representing intermediate results) that are constructed by the query execution engine during the processing of a query and discarded when done.

# Levels of Abstraction

View 1 defines
Faculty(name, department)
based on Professor relation
schema.

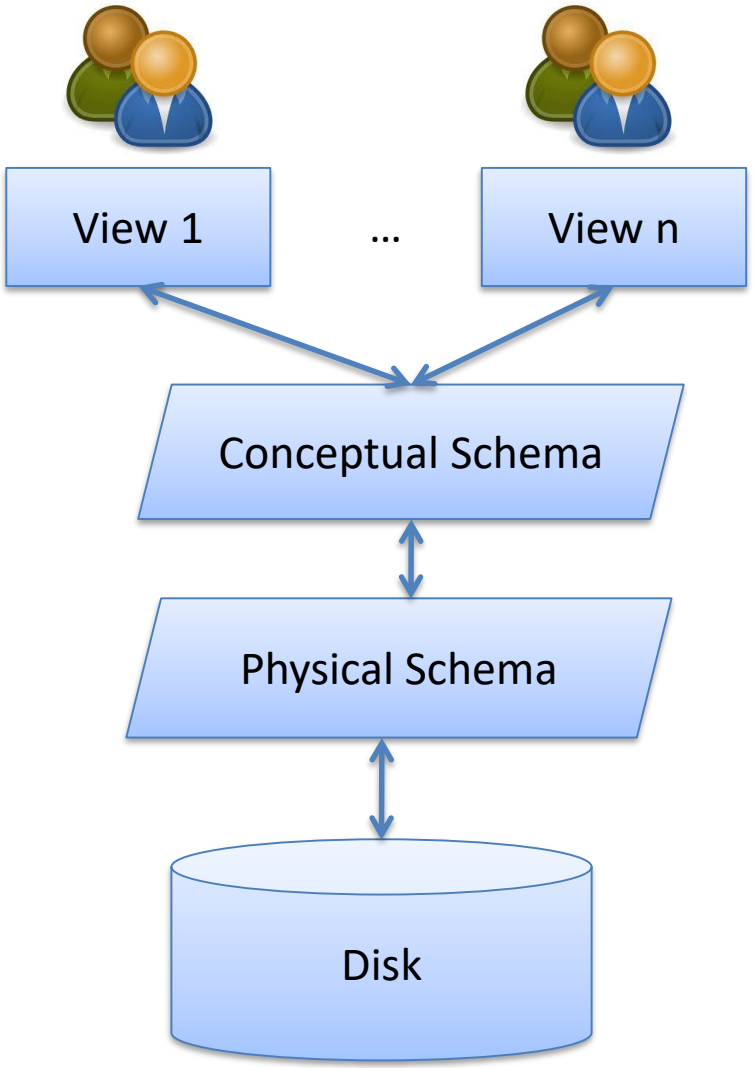
Professor(profid, name, department, salary)

Professor relation may be
stored as an ordered file.



# Two levels of Data Independence

- *Logical data independence*: Protects views from changes in logical (conceptual) structure of data.
  - Which tables do you have?
- *Physical data independence*: Protects conceptual schema from changes in physical structure of data.
  - How are tables stored?



# Physical Data Independence

- The relational data model provides a logical view of the data, and hides the physical representation of data.
  - Data is represented, conceptually, as tables and may not correspond to how it is stored on disk.
  - Operators manipulate data as tables. Users focus formulate queries based on *what* is needed, without knowing *how* the data is stored.
    - Optimizer generates a plan on *how* to compute the answers.
- Advantages:
  - Applications are shielded from low-level details.
  - Physical database design and optimizer can evolve without affecting applications.

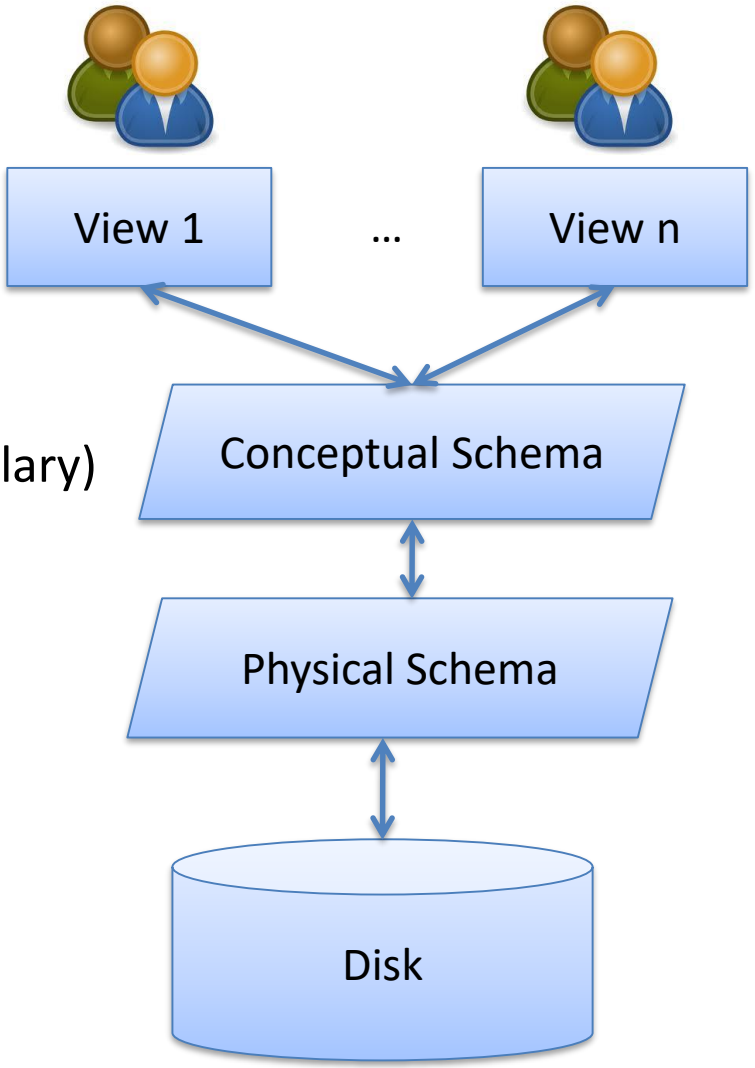
# 1a: Store Professor One Way

View 1 defines
Faculty(name, department)
based on Professor relation
schema.

Professor(profid, name, department, salary)

Professor relation may be
stored as an ordered file.

*Physical data independence*:
Protects conceptual schema
from changes in physical
structure of data.



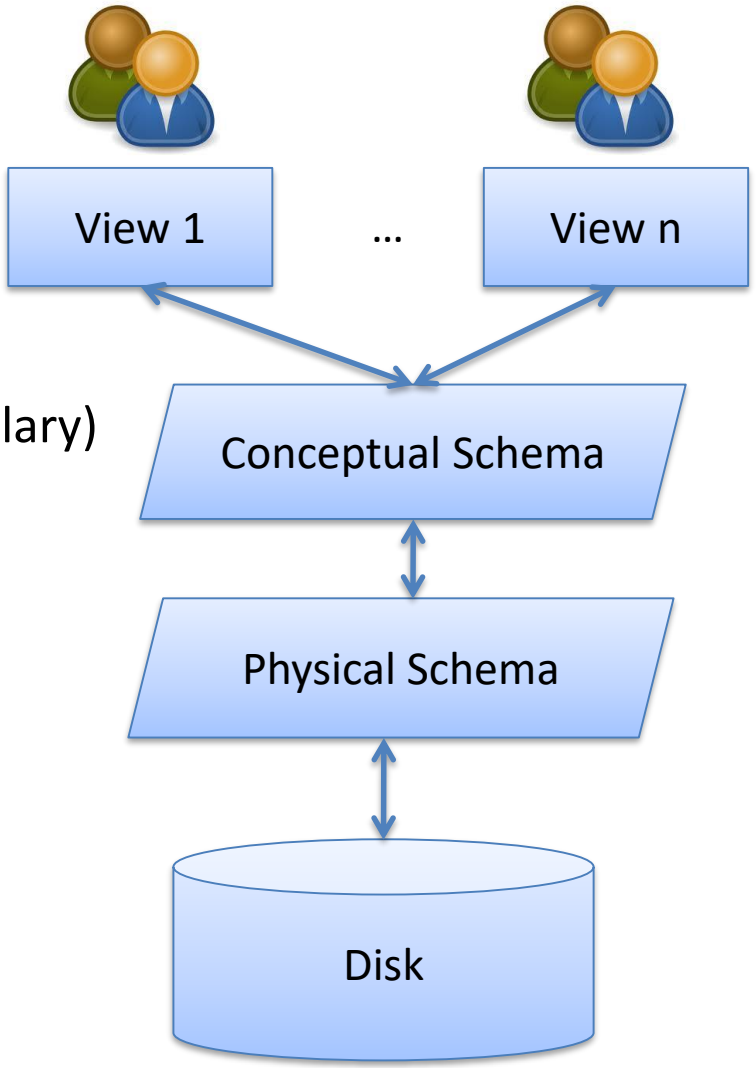
# 1b: Store Professor a Different Way

View 1 defines
Faculty(name, department)
based on Professor relation
schema.

Professor(profid, name, department, salary)

Professor relation may be stored as an unordered file with a B+ tree index on profid.

*Physical data independence*: Protects conceptual schema from changes in physical structure of data.



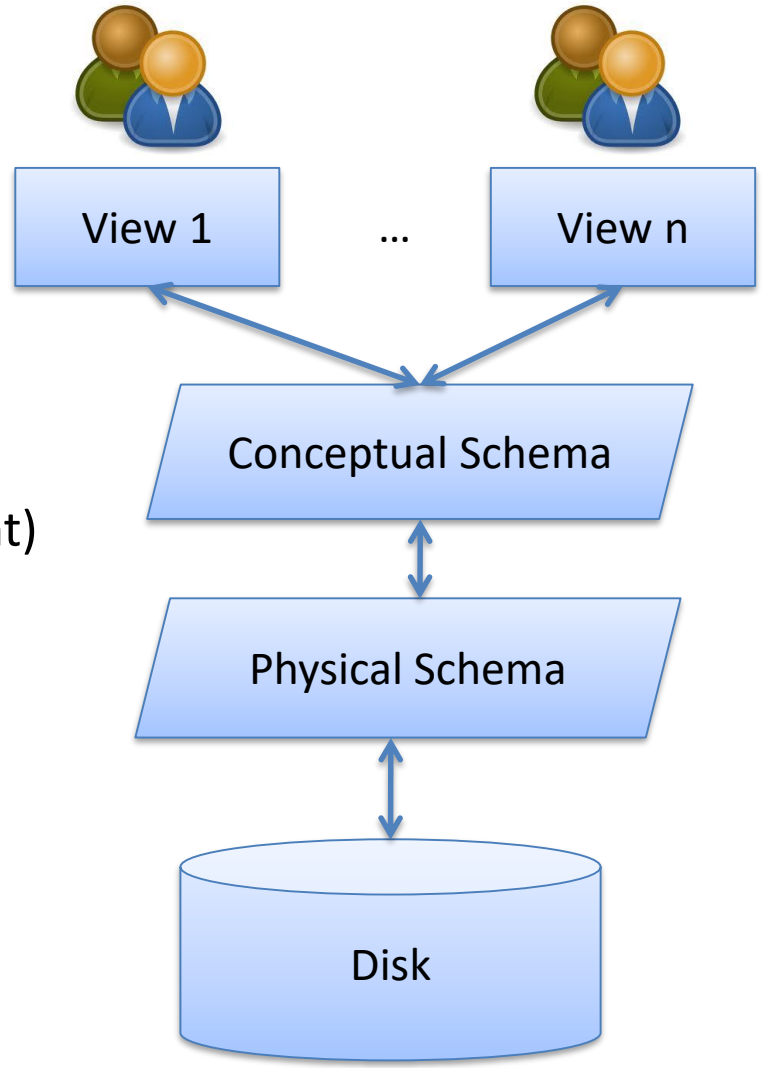
# 2: Change Conceptual Schema

View 1 defines
Faculty(name, department)
based on Professor-public
relation schema. Users
unaffected.

Professor-private(pid, salary)
Professor-public(pid, name, department)

Professor relation may be stored
as an unordered file with a B+
tree index on the pid.

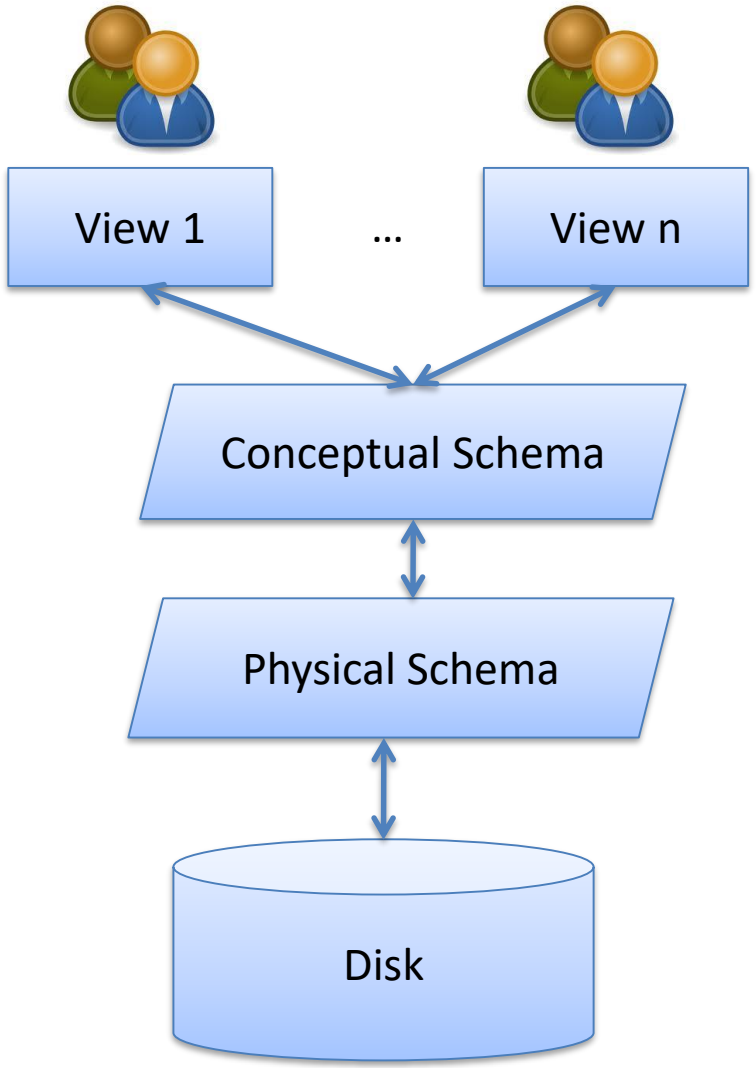
*Logical data independence*:
Protects views from changes
in logical (conceptual)
structure of data.



# Issues for Data Independence

- *Logical data independence*: Protects views from changes in logical (conceptual) structure of data.
  - Okay to change which tables you have, as long as views can be defined correctly to support retrieval and modification operations.

- *Physical data independence*: Protects conceptual schema from changes in physical structure of data.
  - Okay to change physical tables storage.
  - But performance may change depending on how tables are stored!
    - There's an index (or hash) for primary key



# Cartesian Product

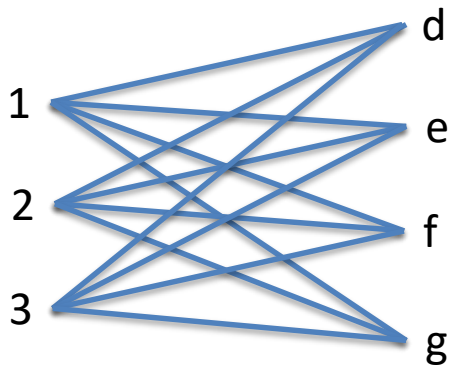
- What is the Cartesian product of {a,b,c,d} and {1,2,3}?

  { (a,1), (a,2), (a,3),
  (b,1), (b,2), (b,3),
  (c,1), (c,2), (c,3),
  (d,1), (d,2), (d,3) }

- What are some examples of relations from that Cartesian product?

# Another Cartesian Product Example

- A: {1,2,3}
- B: {d,e,f,g}
- A × B = { (1,d), (1,e), (1,f), (1,g),
  (2,d), (2,e), (2,f), (2,g),
  (3,d), (3,e), (3,f), (3,g) }




- Suppose that C = {x,y}. What would A x B x C be?

# Math Terminology: Tuples and Relations

- *Tuple*:
  - A *k-tuple* is an ordered sequence of k values (not necessarily different)
    - (1,2) is a binary tuple or 2-tuple
    - (a,b,b) is a ternary tuple or 3-tuple
    - (112, “Ann”, “CS”, “F”, 3.95) is a 5-tuple

- If $D_1$, $D_2$, …, $D_k$ are sets of elements, then the *Cartesian product* $D_1 \times D_2 \times \ldots \times D_k$ is the set of all k-tuples $(d_1, d_2, \ldots, d_k)$ such that $d_i \in D_i$, for all i with $1 \leq i \leq k$.

- *Relation*:
  - A *k-ary relation* is a subset of $D_1 \times D_2 \times \ldots \times D_k$, where each $D_i$ is a set of elements
  - $D_i$ is the *domain (or datatype)* of the *i*th column of the relation
  - In a relational DB, domains may be enumerations, such as {“AMS”, “CMPS”, “TM”}, or may be of standard types (integer, float, date)

# Let's Answer These Questions

Not a homework to be submitted, but you should be able to answer these.

1. If set S is {1,3,5,7} and set T is {2,3,5,7}, what are S UNION T and S INTERSECT T?

2. If set A is {1,2,3} and set B is {u,v,w,x,y}, how many ways can you pick pairs of items, with the first from A and the second from B? (In other words, what is the size of the Cartesian Product A x B?)

3. If you have a set of employees (with names and salaries) where John makes 10K, George makes 20K, Ringo makes 30K and Paul makes 40K, give the names of the employees who make less than the average salary.

4. Write the truth-table for p AND q, where p can be TRUE or FALSE and q can be TRUE or FALSE.